Astrazione e raffinamento

- La soluzione di problemi complessi è ricercata mediante decomposizione del problema e soluzione separata dei sottoproblemi
  - Ciascun sottoproblema viene associato a una opportuna *astrazione*, che consiste in una specifica dei dati e del risultato
  - Ciascun sottoproblema può essere a sua volta analizzato a un livello di *raffinamento* maggiore e decomposto
- I linguaggi di programmazione forniscono *procedure* e *funzioni* per la implementazione di soluzioni ai problemi in termini di astrazioni

  procedura costruisce una istruzione definita dal programmatore; estende la nozione di istruzione

  funzione costruisce una funzione definita dal programmatore; estende la nozione di operatore

Vantaggi della programmazione per astrazioni

facilità di sviluppo Consente di concentrare l'attenzione su un minore numero di aspetti in ogni fase del progetto

chiarezza e facilità di collaudo Il collaudo può essere compiuto separatamente per le soluzioni ai sottoproblemi

possibilità di riutilizzo Il codice può essere riutilizzato piú volte, abbattendo i costi

chiara definizione della comunicazione I dati necessari alla soluzione di un sottoproblema, forniti dal codice chiamante, e i dati modificati dalla soluzione, restituiti al chiamante, sono chiaramente specificati

Parametri delle unità di programma

- Quando, in un programma strutturato in unità, il controllo di esecuzione passa da una unità *chiamante* ad un'altra *chiamata*, i dati necessari alla corretta esecuzione della chiamata e i risultati da essa calcolati sono passati tramite

  variabili non locali: sono variabili visibili sia all'unità chiamante che all'unità chiamata; oppure tramite

  parametri: esiste una *intestazione* della unità chiamata in cui i parametri sono dichiarati;

  - l'unità chiamante collega ai parametri proprie variabili (che possono variare ad ogni attivazione)
  - i parametri dichiarati nella intestazione sono detti *formali*, mentre le variabili collegate in una specifica esecuzione sono i parametri *attuali* o *effettivi*

Legame per valore

- Utilizzato quando la chiamante deve fornire un valore alla chiamata; pertanto la comunicazione è *solo di ingresso* alla chiamata
- Il parametro attuale è una espressione
- Il parametro formale è una variabile locale
- È facoltà della chiamata utilizzare il parametro formale nella sua qualità di variabile. In ogni caso le variabili facenti parte del parametro attuale non mutano di valore.
- È richiesta al compatibilità per assegnamento tra parametro formale e parametro attuale

Legame per riferimento

- Il parametro attuale è una variabile della chiamante
- Il parametro formale si riferisce direttamente alla locazione del parametro attuale; pertanto è un altro nome per il parametro attuale, per tutta e sola la esecuzione della chiamata
- Parametro attuale e formale devono avere lo stesso tipo

Astrazione in C: la funzione

- Il linguaggio C ammette come unico meccanismo di astrazione la *funzione*, costituita da una *intestazione* e un *blocco*
  - Una procedura si realizza come una funzione particolare che non restituisce alcun valore
- Sintassi

$$\begin{aligned}
\langle \text{def-funzione} \rangle &\longrightarrow \langle \text{intest} \rangle \, \langle \text{blocco} \rangle \\
\langle \text{intest} \rangle &\longrightarrow \langle \text{nome-funz} \rangle \, \langle \text{param-formali} \rangle \\
&\quad \mid \langle \text{tipo-risult} \rangle \, \langle \text{nome-funz} \rangle \, \langle \text{param-formali} \rangle \\
\langle \text{tipo-risult} \rangle &\longrightarrow \langle \text{nome-tipo-risult} \rangle \mid \texttt{void} \\
\langle \text{param-formali} \rangle &\longrightarrow \texttt{(} \, \langle \text{lista-param} \rangle \, \texttt{)} \mid \texttt{()} \mid \texttt{(void)} \\
\langle \text{lista-param} \rangle &\longrightarrow \langle \text{param} \rangle \mid \langle \text{param} \rangle \, \texttt{,} \, \langle \text{lista-param} \rangle \\
\langle \text{param} \rangle &\longrightarrow \langle \text{nome-tipo-param} \rangle \, \langle \text{nome-param} \rangle
\end{aligned}$$

Il blocco della funzione

- La parte ⟨dichiarazioni⟩ del blocco è detta *parte dichiarativa locale*; la parte ⟨istruzioni⟩ è detta anche *parte esecutiva*, o *corpo della funzione*
- La parte esecutiva spesso comprende una istruzione `return` con la sintassi

  ⟨istr-return⟩ $\longrightarrow$ **`return`** ⟨espr⟩

  dove ⟨espr⟩ ha il tipo ⟨nome-tipo-risult⟩
- `return` restituisce il risultato della funzione come valore di ⟨espr⟩ e restituisce immediatamente il controllo alla chiamante; pertanto, se è eseguita, è sempre l'ultima istruzione eseguita dalla funzione
- Il valore assunto da ⟨espr⟩ viene comunque convertito a ⟨nome-tipo-risult⟩

Chiamata delle funzioni

- Sintassi

$$\langle \text{chiamata-funzione} \rangle \longrightarrow \langle \text{nome-funz} \rangle \texttt{(} \langle \text{param-attuali} \rangle \texttt{)}$$
$$\mid \langle \text{nome-funz} \rangle \texttt{()}$$
$$\langle \text{param-attuali} \rangle \longrightarrow \langle \text{espr} \rangle \mid \langle \text{espr} \rangle \texttt{,} \langle \text{param-attuali} \rangle$$

- Il collegamento tra parametri attuali e parametri formali
  - è stabilito secondo l'ordine di comparizione
  - è sempre per valore
- $\langle \text{chiamata-funzione} \rangle$ è una $\langle \text{espr} \rangle$: quindi può essere parametro attuale—è consentita la composizione di funzioni

Chiamata delle funzioni

Esempio. Visualizzare il minimo di tre interi acquisiti da input, utilizzando una funzione che calcola il minimo di due interi

```
int MinimoInteri(int i, int j)
{
  if (i<j)
    return i;
  else
    return j;
}
main()
{
  int a,b,c;
  printf("Primo intero="); scanf("%d",&a);
  printf("Secondo intero="); scanf("%d",&b);
  printf("Terzo intero="); scanf("%d",&c);
  printf("%d\n",MinimoInteri(a,MinimoInteri(b,c)));
}
```